Agenda: .NET Introduction

- What is .NET
- Types of Applications we can develop in .NET
- Primary .NET Implementations
- About MS.NET Framework
- Important Components of .NET Framework
- About .NET Implementations
- What is .NET Core
- Benefits of .NET core
- What is new in .NET Core
- .NET Core vs .NET Framework
- About Mono
- About Xamarin
- About Universal Windows Platform (UWP)
- First .NET Core Application.

What is .NET

.NET is a general-purpose development platform. It has several key features, such as support for multiple programming languages, asynchronous and concurrent programming models, and native interoperability, which enable a wide range of scenarios across multiple platforms.

Programming Languages Actively Supported by Microsoft

1. **C#** is simple, powerful, type-safe, and object-oriented, while retaining the expressiveness and elegance of C-style languages.
2. **VB.NET** is an easy language to learn that you use to build a variety of apps that run on .NET. Among the .NET languages, the syntax of VB is the closest to ordinary human language, often making it easier for people new to software development.
3. **F#** is a cross-platform, functional-first programming language that also supports traditional object-oriented and imperative programming.

Development Tools

1. Visual Studio.NET.
2. VS Code.

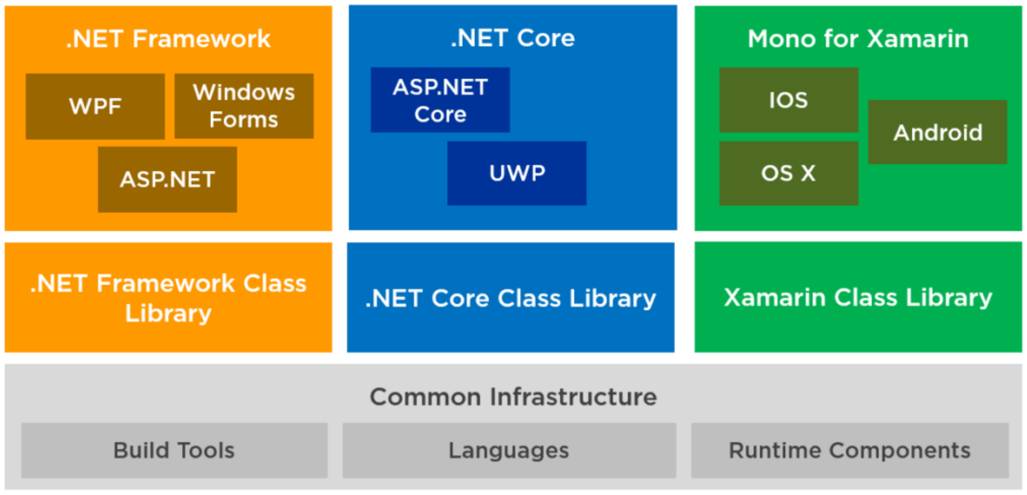
Key Features of .NET

1. Multiple Programming Language
2. Automatic Memory Management
3. Generics
4. Type Safety
5. LINQ
6. Native Interoperability
7. Async Programming

Primary .NET Implementations

Following are **primary .NET implementations** that Microsoft actively develops and maintains:

1. .NET Framework
2. Mono
3. Xamarin
4. .NET Core



Each implementation of .NET includes the following components:

- **One or more runtimes**. Examples: CLR for .NET Framework, CoreCLR and CoreRT for .NET Core.
- A class library that implements the **.NET Standard** and may **implement additional APIs**. Examples: .NET Framework Base Class Library, .NET Core Base Class Library.
- Optionally, one or more **application frameworks**. Examples: ASP.NET, Windows Forms, and Windows Presentation Foundation (WPF) are included in the .NET Framework.
- Optionally, **development tools**. Some development tools are shared among multiple implementations.

Types of Applications we can develop in .NET

1. Console Based Applications

2. Windows Application (GUI)
    a. WinForms
    b. Windows Presentation Foundation (WPF)
    c. UWP - Universal Windows Platform Application
3. Windows Services
4. Web Based Applications
    a. Web Forms (ASPX / AMSX)
    b. MVC
    c. Razor Pages
5. Service Oriented Applications
    a. ASP.NET ASMX
    b. SOAP Services using Windows Communication Framework (WCF)
    c. RESTful Services using WFC and ASP.NET Web API
6. Mobile Applications
    a. Xamarin for Native App development for Android, IOS and Windows Phone
    b. Apache Cordova for Hybrid App Development

.NET Runtimes:

A runtime is the execution environment for a managed program. The OS is part of the runtime environment but is not part of the .NET runtime. Here are some examples of .NET runtimes:

- Common Language Runtime (CLR) for the .NET Framework
- Core Common Language Runtime (CoreCLR) for .NET Core
- .NET Native for Universal Windows Platform
- The Mono runtime for Xamarin.iOS, Xamarin.Android, Xamarin.Mac, and the Mono desktop framework

.NET Compilers:

- **Roslyn** is a compiler platform and includes the C# and VB compilers and other tools. These compilers emit **Common Intermediate Language** (CIL) code. It provides open-source C# and Visual Basic compilers with rich code analysis APIs. It enables building code analysis tools with the same APIs that are used by Visual Studio. Roslyn produces platform independent Intermediate Language (IL) and is used when building against .NET 2015, including Framework and Core. At release, the entire .NET Framework will be compiled using Roslyn.
- **RyuJIT** is the new default just-in-time (JIT) compiler for .NET on x64. The JIT compiler takes IL and compiles it for the particular machine architecture the first time it is executed at run-time. Used for desktop and server-based scenarios, RyuJIT is an overhaul of the previous 64-bit JIT compiler that significantly reduces startup times. It also includes

support for SIMD (single instruction, multiple data) which allows mathematical operations to execute over a set of values in parallel.

- **.NET Native** compiles C# and VB to native machine code that performs like C++, so developers continue to benefit from the productivity and familiarity of the .NET Framework with the performance of native code. Typically, apps that target .NET are compiled to intermediate language (IL). At run time, the just-in-time (JIT) compiler translates the IL to native code. In contrast, .NET Native is an ahead-of-time compiler that compiles apps directly to native code and contains a minimal CLR runtime. Popular Windows Store apps start up to 60% faster and use 15-20% less memory when compiled with .NET Native. Universal Windows apps will run on .NET Native (ARM, x86, x64). See: Compiling Apps with .NET Native

About MS.NET Framework

The first .NET implementation that has existed since 2002.

Its best optimized for developing Windows desktop applications using WinForms and WPF.

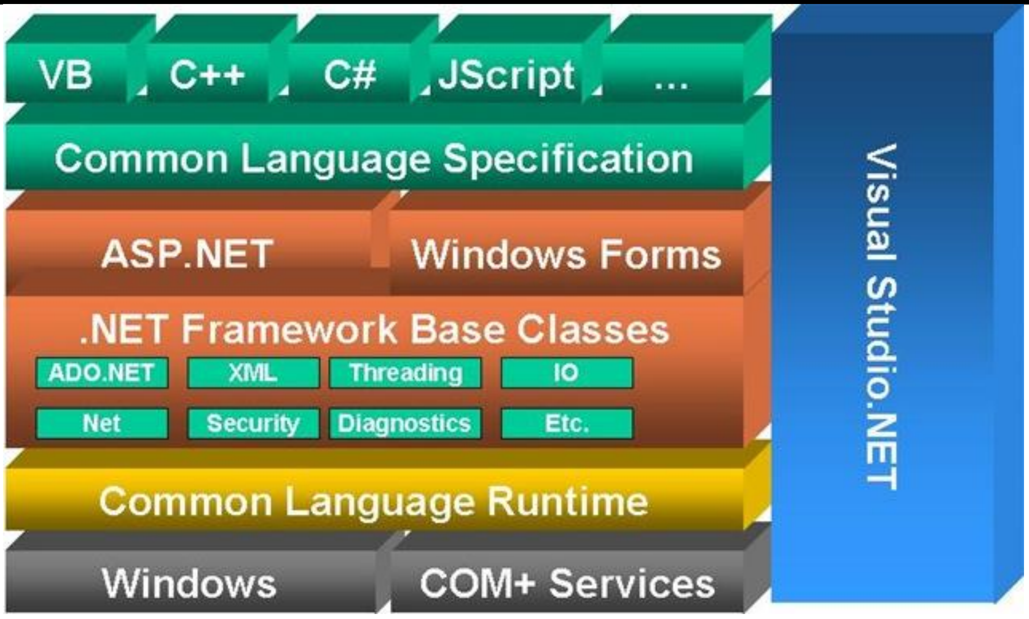
It supports the largest set of APIs, you’re much less likely to have to worry about whether you can easily accomplish some complex task if you target the .NET Framework.

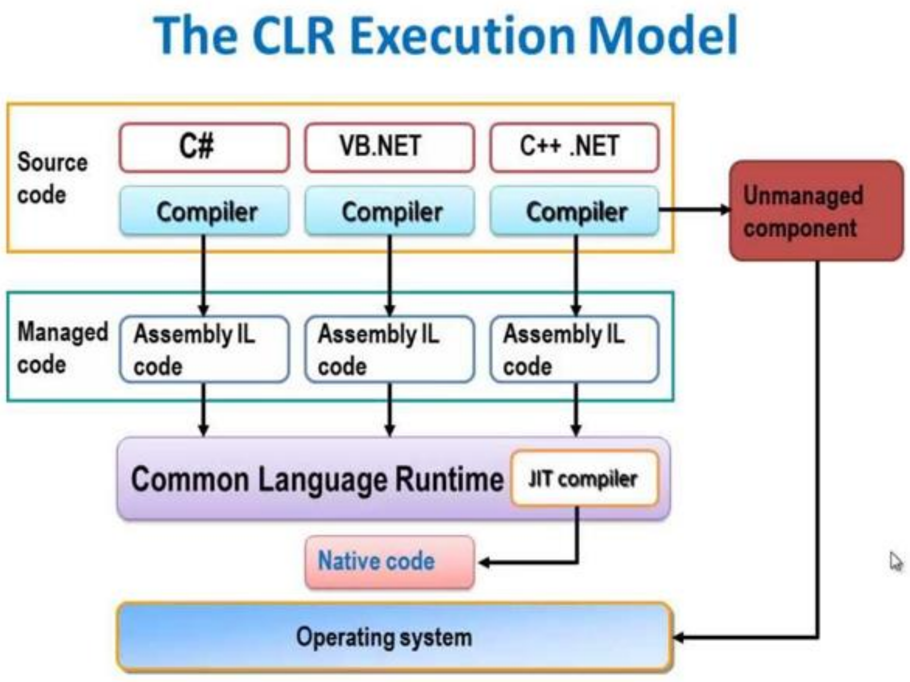
Version History of .NET Framework

Version number	CLR version	Release date	Support ended	Development tool	Replaces
1.0	1.0	2002-02-13	2009-07-14	Visual Studio .NET	N/A
1.1	1.1	2003-04-24	2015-06-14	Visual Studio .NET 2003	1.0
2.0	2.0	2005-11-07	2011-07-12	Visual Studio 2005	N/A
3.0	2.0	2006-11-06	2011-07-12	Expression Blend	2.0
3.5	2.0	2007-11-19	N/A	Visual Studio 2008	2.0, 3.0
4.0	4	2010-04-12	2016-01-12	Visual Studio 2010	N/A
4.5	4	2012-08-15	2016-01-12	Visual Studio 2012	4.0
4.5.1	4	2013-10-17	2016-01-12	Visual Studio 2013	4.0, 4.5
4.5.2	4	2014-05-05	N/A	N/A	4.0–4.5.1
4.6	4	2015-07-20	N/A	Visual Studio 2015	4.0–4.5.2
4.6.1	4	2015-11-30	N/A	Visual Studio 2015 Update 1	4.0–4.6
4.6.2	4	2016-08-02	N/A		4.0–4.6.1
4.7	4	2017-04-05	N/A	Visual Studio 2017	4.0–4.6.2
4.7.1	4	2017-10-17	N/A	Visual Studio 2017	4.0–4.7
4.7.2	4	2018-04-30	N/A	Visual Studio 2017	4.0–4.7.1
4.7.3	4	Developing	N/A	Visual Studio 2017	4.0–4.7.2

4.8	4	Developing	N/A	Visual Studio 2019 (Planning)	4.0–4.7.3

Important Components of .NET Framework





Quick overview of Few Keywords or Important Terms

1. CIL Code = Common Intermediate Language Code

2. Framework Class Libraries
3. Assembly
4. Common Type System
5. Common Language Runtime
6. Garbage Collector
7. JIT Compiler

Mono

- Mono, the open source and free development platform based on the .NET Framework, allows developers to build cross-platform applications with improved developer productivity.
- Was launched on July 19, 2001, after Microsoft first announced their .NET Framework in June 2000
- The project was started by enthusiasts, led by **Miguel de Icaza**, who believed that the benefits of .NET should be enjoyed on other platforms besides Windows, and that the best way to accomplish this was through an open-source effort.
- The supervision of Mono has moved from one company to another, as **de Icaza** moved:
  Ximian → Novell → Xamarin → Microsoft.

It can run ASP.NET, ADO.NET, Silverlight and Windows.Forms applications without recompilation
The easiest way to describe what Mono currently supports is:
Everything in .NET 4.7 except **WPF**, **WWF**, and with **limited WCF** and **limited ASP.NET async stack**
Mono also powers games built using the Unity engine.

Supported Platforms: Mono has support for both 32 and 64 bit systems on a number of architectures as well as a number of operating systems.

- Linux
- macOS, iOS, tvOS, watchOS
- Sun Solaris
- IBM AIX and i
- BSD - OpenBSD, FreeBSD, NetBSD
- Microsoft Windows
- Sony PlayStation 4
- XboxOne

Mono Components

- **C# Compiler** - Mono's C# compiler is feature complete for C# 1.0, 2.0, 3.0, 4.0, 5.0 and 6.0 (ECMA). A good description of the feature of the various versions is available on Wikipedia.
- **Mono Runtime** - The runtime implements the ECMA Common Language Infrastructure (CLI). The runtime provides a Just-in-Time (JIT) compiler, an Ahead-of-Time compiler (AOT), a library loader, the garbage collector, a threading system and interoperability functionality.
- **.NET Framework Class Library** - The Mono platform provides a comprehensive set of classes that provide a solid foundation to build applications on. These classes are compatible with Microsoft's .Net Framework classes.
- **Mono Class Library** - Mono also provides many classes that go above and beyond the Base Class Library provided by Microsoft. These provide additional functionality that are useful, especially in building Linux applications. Some examples are classes for Gtk+, Zip files, LDAP, OpenGL, Cairo, POSIX, etc.

Languages Supported on Mono:
C#, VB.NET, F#, Java, Python, JavaScript, PHP, Object Pascal, Lua, Cobra, Synergy-DBL, Scala, Nemerle
https://www.mono-project.com/docs/about-mono/languages/

MonoDevelop is a Mono/C# Integrated Development Environment (IDE) for Windows, Linux and macOS. It parses your C# and VB programs as you type them and is able to provide contextual completion of methods.
You can continue to use **Visual Studio** to develop your applications on Windows, the binaries produced by Visual Studio are binary compatible with Mono, so you only need to get these files to your Linux/Unix server.

In summary, if you have a .NET Framework application for Windows, and you now want your application to support Windows, macOS, Linux, BSD, and a bunch of other platforms but you want to spend a minimal amount of effort on converting your .NET application to run on many platforms, Mono may still be your best bet.

Microsoft now seems to be focusing most of their Mono efforts on the **Xamarin platform and support for iOS and Android**, although Mono support for desktop and server platforms seems to be active, too.

Xamarin

Xamarin is now a Microsoft-owned software company founded in May 2011 by the engineers that created Mono, Mono for Android and MonoTouch, which are cross-platform implementations of Microsoft .NET specifically for mobile products. With a C#-shared codebase, developers can use Xamarin tools to write native Android, iOS, and Windows apps with native user interfaces and share code across multiple platforms, including Windows and macOS.
According to Xamarin, over 1.4 million developers were using Xamarin's products in 120 countries around the world as of April 2017.
On February 24, 2016, Microsoft announced it had signed a definitive agreement to acquire Xamarin.

Xamarin offers two commercial products: **Xamarin.iOS** and **Xamarin.Android**. They're both built on top of *Mono*, an open-source version of the .NET Framework.

On iOS, Xamarin's *Ahead-of-Time* (*AOT*) Compiler compiles Xamarin.iOS applications directly to native ARM assembly code. On Android, Xamarin's compiler compiles down to *Intermediate Language* (*IL*), which is then *Just-in-Time* (*JIT*) compiled to native assembly when the application launches.

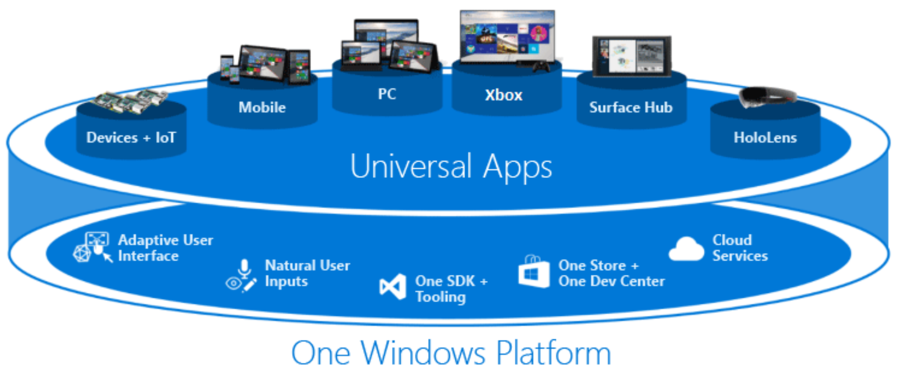
Application Output

When Xamarin applications are compiled, the result is an Application Package, either an **.app** file in iOS, or .**apk** file in Android. These files are indistinguishable from application packages built with the platform's default IDEs and are deployable in the exact same way.

Universal Windows Platform (UWP)

UWP is an implementation of .NET that is used for building modern, touch-enabled Windows applications and software for the Internet of Things (IoT). It's designed to unify the different types of devices that you may want to target, including PCs, tablets, phablets, phones, and even the Xbox.

UWP are developed and published into Microsoft Store for distribution.



Apps can be written in C++, C#, VB.NET, and JavaScript. For UI, use XAML, HTML, or DirectX.

When using C# and VB.NET, the .NET APIs are provided by .NET Core.

Able to use a common API on all devices that run Windows 10.

Able to use device specific capabilities and adapt the UI to different device screen sizes, resolutions, and DPI.

Available from the Microsoft Store on all devices (or only those that you specify) that run on Windows 10. The Microsoft Store provides multiple ways to make money on your app.

What is .NET Core?

- .NET Core is a **general-purpose development** platform maintained by Microsoft and the .NET community on GitHub.
- It is free and **open source cross-platform**, supporting **Windows, macOS and Linux.** It was contributed to **.NET Foundation** by Microsoft in 2014 and is now most activate .NET Foundation project.
- On Linux, Microsoft primarily supports .NET Core running on **Red Hat Package Manger** (RPM) and **Debian** distribution families (Ubuntu / Linux Mint).
- **Consistent across architectures:** Runs your code with the same behavior on multiple architectures, including x64, x86, and ARM.

As of today, .NET Core supports following app models:

- Console apps
- ASP.NET Core apps and services
- .NET Core Libraries
- Universal Windows Platform (UWP) apps.

- It includes easy-to-use **command-line tools** that can be used for local development and in continuous-integration scenarios.
- It provides **flexible deployment** and can be included in your app or installed side-by-side user- or machine-wide. Can be used with Docker containers.
- .NET Core provides compatibility with .NET Framework and Mono APIs by implementing the .NET Standard specification.
- **Roslyn** is a compiler platform and includes the C# and VB compilers and other tools. These compilers emit **Common Intermediate Language** (CIL) code.
- **CoreCLR:** A complete **runtime implementation of CLR** where CIL code is compiled into machine code using a JIT compiler while tha app is running. It's the **virtual machine** that manages the execution of .NET programs. CoreCLR started as a **copy of CLR**. It has been modified to support **different OSes**. They're maintained separately and in parallel.

What is Core CLR

Its platform specific runtime implementation of .NET Core.

Source Code of .NET Core (C#) => Roslyn Compiler => CIL Code (Platform Independent) (Binary form of CIL = EXE / DLL – Portable Executable)

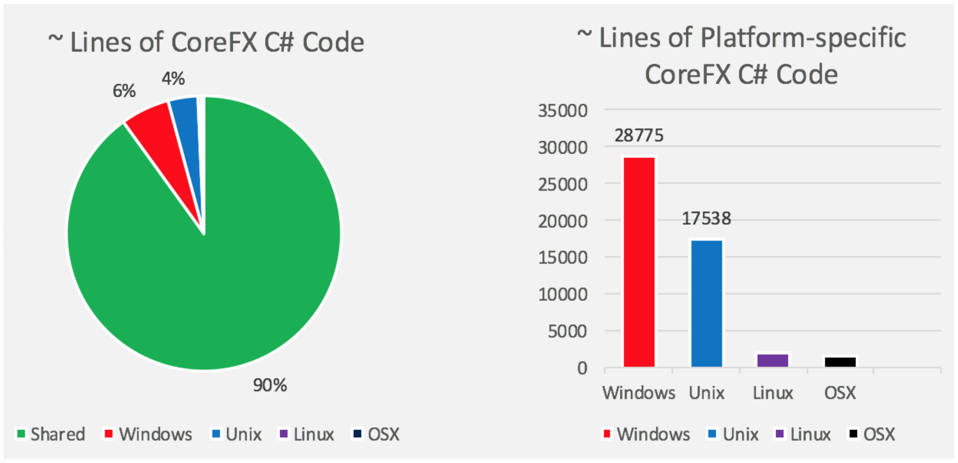
- CIL Code on Ubuntu = CoreCLR of Ubuntu
- CIL Code on Windows = CoreCLR or Windows
- CIL Code on MacOS = CoreCLR on Mac

What is CoreFX (.NET Core Libraries)

- **CoreFX** is a platform neutral code that is shared across all platforms.
- It comprises of **framework libraries**, which provide primitive data types, app composition types and fundamental utilities example System.Collections, System.IO, System.Xml, and many other components.

.NET Core and CoreFX

- While .NET Core shares a subset of .NET Framework APIs, it comes with its own API that is not part of .NET Framework.
- .NET Core includes CoreFX, which is a partial fork of FCL.
- CoreFX is a mix of **platform-specific and platform-neutral** libraries in .NET Core. You can see the pattern in a few examples:
    - CoreCLR is **platform-specific**. It's built in C/C++, so is platform-specific by construction. It builds on top of OS subsystems, like the memory manager and thread scheduler.
    - System.IO and System.Security.Cryptography.Algorithms are **platform-specific**, given that the storage and cryptography APIs differ significantly on each OS.
    - System.Collections and System.Linq are **platform-neutral**, given that they create and operate over data structures.
    - Chart below shows clearly the vast majority of **CoreFX is platform-neutral code** that is shared across all platforms. Platform-neutral code can be implemented as a single portable assembly that is used on all platforms.



Windows and Unix implementations are similar in size. Windows has a larger implementation since CoreFX implements some Windows-only features, such as **Microsoft.Win32.Registry** but **does not yet** implement any Unix-only concepts.

Use .NET Core when:

1. There are Cross platform needs.
2. Microservices are being used.

3. Docker containers are being used.
4. Applications needs high performance and scalability.
5. If you want CLI control.

Not to use .NET Core when:

1. For Windows Forms or WPF applications.
2. ASP.NET WebForms.
3. WCF Services.
4. You need access to Windows specific API's like Windows Registry, WMI etc

Version History of .NET Core

- .NET Core 1.0 was released on **27 June 2016,** along with Visual Studio 2015 Update 3
- .NET Core 1.0.4 and .NET Core 1.1.1 were released along with .NET Core Tools 1.0 and **Visual Studio 2017** on **7 March 2017.**
- .NET Core 2.0 was released on **14 August 2017** along with **Visual Studio 2017 15.3,** ASP.NET Core 2.0, and Entity Framework Core 2.0.
- .NET Core 2.1 was released on **30th May 2018** along with **Visual Studio 2017 15.7** and will be a **long-term support** (LTS) release. This means that it is supported for three years. Microsoft recommends that you make .NET Core 2.1 as your new standard for .NET Core development.
- **.NET Core 2.2** was released on **04th Dec 2018** along with **Visual Studio 2017 15.9.** It includes diagnostic improvements to the runtime, support for ARM32 for Windows and Azure Active Directory for SQL Client.
- .NET Core 3 was announced on **May 7, 2018**, at Microsoft Build. A public **preview** was released on December 4, 2018. An official release is planned for 2019. With .NET Core 3 the framework will get support for development of **desktop application software, artificial intelligence/machine learning and IoT apps**.

Platform Support: .NET Core 2.1 is supported on the following operating systems:

- Windows Client: 7, 8.1, 10 (1607+)
- Windows Server: 2008 R2 SP1+
- macOS: 10.12+
- RHEL: 6+
- Fedora: 26+
- Ubuntu: 14.04+
- Debian: 8+
- SLES: 12+
- openSUSE: 42.3+

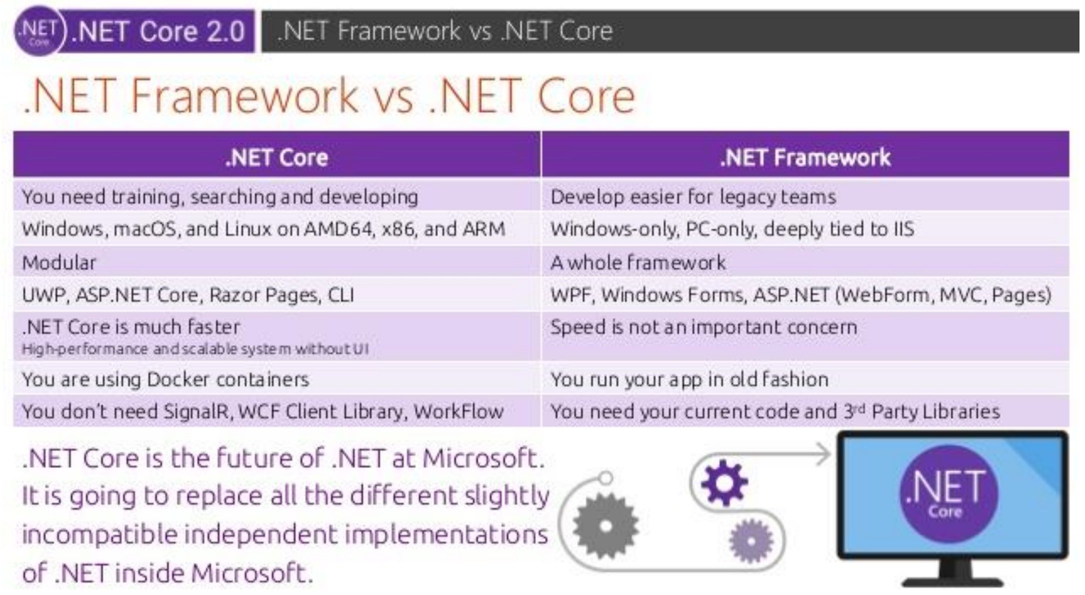
- Alpine: 3.7+

Chip support:

- x64 on Windows, macOS, and Linux
- x86 on Windows
- ARM32 on Linux (Ubuntu 18.04+, Debian 9+)

Note: .NET Core 2.1 is supported on **Raspberry Pi 2+**. It isn't supported on the Pi Zero or other devices that use an ARMv6 chip. .NET Core requires ARMv7 or ARMv8 chips.

.NET Core vs .NET Framework vs Mono


.NET Core 2.0
.NET Framework vs .NET Core

.NET Framework vs .NET Core

.NET Core	.NET Framework
You need training, searching and developing	Develop easier for legacy teams
Windows, macOS, and Linux on AMD64, x86, and ARM	Windows-only, PC-only, deeply tied to IIS
Modular	A whole framework
UWP, ASP.NET Core, Razor Pages, CLI	WPF, Windows Forms, ASP.NET (WebForm, MVC, Pages)
.NET Core is much faster High-performance and scalable system without UI	Speed is not an important concern
You are using Docker containers	You run your app in old fashion
You don't need SignalR, WCF Client Library, WorkFlow	You need your current code and 3rd Party Libraries

.NET Core is the future of .NET at Microsoft. It is going to replace all the different slightly incompatible independent implementations of .NET inside Microsoft.

- **App-models**: .NET Core does not support all the .NET Framework app-models, because many of them are built on Windows technologies, such as WPF (built on top of DirectX). The **Console and ASP.NET Core** app-models are supported by both .NET Core and .NET Framework.
- **APIs**: .NET Core contains many of the same, but fewer, APIs as the .NET Framework, and with a different factoring (assembly names are different). These differences require changes to source to port code to .NET Core (APIPort tool). .NET Core implements the **.NET Standard API**, which will grow to include more of the .NET Framework BCL API over time.
- **NuGet packages:** .NET Core is also modular, meaning that instead of assemblies, developers work with **NuGet packages**. Unlike .NET Framework, which is serviced using Windows Update, .NET Core relies on its package manager to receive updates.

- **Subsystems** -- .NET Core implements a subset of the subsystems in the .NET Framework, with the goal of **a simpler implementation and programming model**. For example, Code Access Security (CAS) is not supported, while reflection is supported.
- **Platforms:** The .NET Framework supports Windows and Windows Server while .NET Core also supports macOS and Linux.
- **Open Source:** .NET Core is open source, while a read-only subset of the .NET Framework is open source.

Comparison with Mono

Mono is the original **cross-platform** and open source .NET implementation, first shipping in 2004. It can be thought of as a **community clone** of the .NET Framework. The Mono project team relied on the open .NET standards published by Microsoft in order to provide a compatible implementation.

The major differences between .NET Core and Mono:

- **App-models** -- Mono supports a subset of the .NET Framework app-models (for example, Windows Forms) and some additional ones (for example, Xamarin.iOS) through the Xamarin product. .NET Core doesn't support these.
- **APIs** -- Mono supports a large subset of the .NET Framework APIs, using the same assembly names and factoring.
- **Platforms** -- Mono supports many platforms and CPUs.
- **Open Source** -- Mono and .NET Core both use the MIT license and are .NET Foundation projects.
- **Focus** -- The primary focus of Mono in recent years is mobile platforms, while .NET Core is focused on cloud and desktop workloads.

Hello World .NET Core Application

URL to download: Download Link: https://dotnet.microsoft.com/download

Verify the installation

```
C:\>dotnet --version
C:\>dotnet --info
```

Steps to Build Hello World Application

1. Create a Folder "HelloWorldDemo" in any location
2. Type the following command

   dotnet new console

   This command creates Program.cs file in your folder with simple "Hello World" program

Run the "HelloWorldDemo" program

3. Type the following command in terminal

   dotnet build

dotnet run

Debug the application

4. Open *Program.cs* by clicking on it. The first time you open a C# file in Visual Studio Code, **OmniSharp** loads in the editor.
5. Visual Studio Code should prompt you to add the missing assets to build and debug your app. Select **Yes**.
6. To open the Debug view, click on the Debugging icon on the left side menu.
7. Locate the green arrow at the top of the pane. Make sure the drop-down next to it has **.NET Core Launch (console)** selected.
8. Add a breakpoint to your project by clicking on the **editor margin**, which is the space on the left of the line numbers in the editor, next to line 9.
9. To start debugging, select F5 or the green arrow.

Note: Starting with .NET Core, you don't have to run **dotnet restore** because it's run implicitly by all commands that require a restore to occur, such as **dotnet new**, **dotnet build** and **dotnet run**. It's still a valid command in certain scenarios where doing an explicit restore makes sense, such as continuous integration builds in Visual Studio Team Services or in build systems that need to explicitly control the time at which the restore occurs.

Command to generate the project in **VB.NET**

dotnet new console -lang vb